Solving the Tree Containment Problem for Genetically Stable Networks in Quadratic Time

Philippe Gambette[1], Andreas D. M. Gunawan[2], Anthony Labarre[1], Stéphane Vialette[1], and Louxin Zhang[2]

[1] Université Paris-Est, LIGM (UMR 8049), UPEM, CNRS, ESIEE, ENPC, F-77454, Marne-la-Vallée, France
[2] Department of Mathematics, National University of Singapore

Abstract. A phylogenetic network is a rooted acyclic digraph whose leaves are labeled with a set of taxa. The tree containment problem is a fundamental problem arising from model validation in the study of phylogenetic networks. It asks to determine whether or not a given network displays a given phylogenetic tree over the same leaf set. It is known to be NP-complete in general. Whether or not it remains NP-complete for stable networks is an open problem. We make progress towards answering that question by presenting a quadratic time algorithm to solve the tree containment problem for a new class of networks that we call genetically stable networks, which include tree-child networks and comprise a subclass of stable networks.

1 Introduction

With thousands of genomes being fully sequenced, phylogenetic networks have been adopted to study "horizontal" processes that transfer genetic material from a living organism to another without descendant relation. These processes are a driving force in evolution which shapes the genome of a species [1, 9].

A *rooted* (*phylogenetic*) *network* over a set X of taxa is a rooted acyclic digraph with a set of leaves (*i.e.*, vertices of outdegree 0) that are each labeled with a distinct taxon. Such a network represents the evolutionary history of the taxa in X, where the *tree nodes* (*i.e.*, nodes of indegree 1) represent speciation events. The nodes of indegree at least two are called *reticulations* and represent genetic material flow from several ancestral species into an "unrelated" species. A plethora of methods for reconstructing networks and related algorithmic issues have been extensively studied over the past two decades [4, 5, 8, 10].

One of the ways of assessing the quality of a given phylogenetic network is to verify that it is consistent with previous biological knowledge about the species. Biologists therefore demand that the network display existing gene trees, and the corresponding algorithmic problem is known as the *tree containment* problem (or TC problem for short) [5], which is well-known to be NP-complete [7, 6]. Great efforts have been devoted to identifying tractable subclasses of networks, such as binary galled trees [7], normal networks, binary tree-child networks, level-k

networks [6], or nearly-stable networks [3]. One of the major open questions in this setting is the complexity of the TC problem on the so-called *stable networks*.

A node v in a network is *stable* if there exists a leaf such that every path from the root to the leaf passes through v. A network is *stable* (or *reticulation visible*) [5] if every reticulation is stable. Motivated by the study in [2], we make progress in this work towards determining the complexity of the TC problem on stable networks by presenting a quadratic-time algorithm for a new class that we call *genetically stable networks*. As we shall show, these networks comprise a subclass of stable, tree-sibling networks, including *tree-child* networks.

2 Concepts and Notions

2.1 Binary networks

We focus in this paper on *binary* networks, *i.e.* networks whose root has indegree 0 and outdegree 2, whose internal nodes all have degree 3, and whose leaves all have indegree 1 and outdegree 0. An internal node in a network N is called a *tree node* if its indegree and outdegree are 1 and 2, respectively. It is called a *reticulation* (*node*) if its indegree and outdegree are 2 and 1, respectively. A node v is said to be *below* a node u if u is an ancestor of v, *i.e.* there is a directed path from u to v in N.

We also assume that in a binary network, there is a path from its root to every leaf and that a node can be of indegree 1 and outdegree 1. We also draw an open edge entering the root so that the root becomes a tree node with degree 3, as shown in Figure 1. For a network or a subnetwork N, we use the following notation: $\rho(N)$ for its root, $\mathcal{L}(N)$ for its leaf set, $\mathcal{R}(N)$ for the set of reticulations, $\mathcal{T}(N)$ for the set of tree nodes, $\mathcal{V}(N)$ for its vertex set (*i.e.*, $\mathcal{R}(N) \cup \mathcal{T}(N) \cup \mathcal{L}(N) \cup \{\rho(N)\}$), $\mathcal{E}(N)$ for its edge set, $p(u)$ for the set of parents of $u \in \mathcal{R}(N)$ or the unique parent of u otherwise, children(u) for the set of children of $u \in \mathcal{T}(N)$ or the unique child of $u \in \mathcal{R}(N)$, and $\mathcal{P}_N(u, v)$ for the set of all paths from a node u to a node v in N.

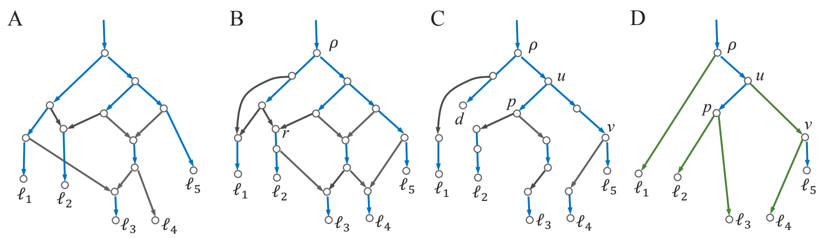

Fig. 1. (**A**) A nearly tree-child network. (**B**) A non-nearly tree-child network, in which the parents of r are not connected to any leaf by a tree path. (**C**) A subtree T' obtained by removing an incoming edge from each reticulation in the network in **B**. (**D**) A tree obtained from T' by contraction.

A path P from u to v in a network is a *tree path* if every internal node of P, that is every node in $\mathcal{V}(P) \setminus \{u, v\}$, is a tree node. For a network N and an edge subset $E \subseteq \mathcal{E}(N)$, $N - E$ denotes the subnetwork with vertex set $\mathcal{V}(N)$ and edge set $\mathcal{E}(N) - E$. For a node subset $S \subset \mathcal{V}(N)$, $N - S$ denotes the subnetwork with vertex set $\mathcal{V}(N) - S$ and edge set $\{(u, v) \in \mathcal{E}(N) \mid u \notin S, v \notin S\}$. When E or S has only one element x, we simply write $N - x$. A leaf in the resulting network is a dummy leaf if it is not a leaf in the original network N.

2.2 The Tree Containment (TC) Problem

Let N be a binary network and T a binary tree over the same set of taxa. We say that N *displays* T if N contains a subtree T', obtained by removing an incoming edge for each reticulation in N, such that T can be obtained from T' by:

1. recursively removing dummy leaves (such as d in Figure 1.C), and
2. contracting every path containing only nodes of degree 2 into a single edge (Figure 1.B-D).

T' is then referred to as a *subdivision* of T in N. Given a binary network and a binary tree, the *tree containment (TC) problem* is to determine whether or not the network displays the tree [5]. This problem is known to be NP-complete [7, 6], and a large part of the current research therefore focuses on finding tractable classes of binary networks that are as general as possible.

3 Genetically Stable Networks

Let N be a binary network and $u, v \in \mathcal{V}(N)$. Node u is *stable on node v* if every path from $\rho(N)$ to v passes through u. We denote by $\mathcal{PDL}_N(u)$ the set of leaves on which u is stable, and say that u is *stable* (or *visible*) if $\mathcal{PDL}_N(u) \neq \emptyset$. Network N is itself *stable* if every $r \in \mathcal{R}(N)$ is stable. The network in Figure 1.A is stable, whereas the one in Figure 1.B is not. The following result will be useful.

Proposition 1. *Let N be a binary network and $r \in \mathcal{R}(N)$ with $p(r) = \{u, v\}$.*

(a) *If $s \in \mathcal{V}(N)$ is a stable node, then* children(s) *contains a tree node.*
(b) *If N is stable, then both parents of each reticulation are tree nodes.*
(c) *For any descendant x of r, either u or v is not stable on x.*
(d) *If r and u are stable on the same leaf, then u is stable on v.*
(e) *If r is stable on $\ell \in \mathcal{L}(N)$ and v is stable on $\ell' \in \mathcal{L}(N)$ but not on ℓ, then u is not in a path from v to ℓ'. Additionally, there is no z in a path from v to ℓ' that is connected to u by a tree path.*

If a tree node is stable on a leaf ℓ, then its unique parent is also stable on ℓ, but the stability of a reticulation does not imply that of its parents. Cordue, Linz and Semple [2] recently introduced a class of stable networks that we call *nearly tree-child networks* and which satisfy the property that every reticulation has a parent connected to some leaf by a tree path (see Figure 1.A for an example).

In this paper, we are interested in stable networks in which every reticulation has a stable parent. We coin the concept *genetic stability* (GS) to describe such networks, which conveys the idea that each reticulation inherits its stability from one of its parents. Note that in a nearly tree-child network, there is a tree path from one of the parents of every reticulation to a leaf, so that parent must be stable. On the other hand, a GS network may not be a nearly tree-child network[3]. Therefore, GS networks comprise a proper superclass of the nearly tree-child networks.

A network is *tree-sibling* if every reticulation has at least one sibling that is a tree node [5]. Interestingly, we also have the following fact.

Proposition 2. *Every GS network is tree-sibling.*

Our result on the complexity of the TC problem for binary GS networks therefore refines the complexity gap of the TC problem between the classes of binary tree-child networks, where it can be solved in polynomial time, and tree-sibling networks where it is NP-complete [6]. Furthermore, a study of the properties of networks simulated using the coalescent model with recombination shows that the percentage of simulated networks which are GS is significantly larger than that of tree-child networks (see `http://phylnet.info/recophync/`), thus making that new class significant in practice.

4 Solving the TC Problem For GS Networks

In this section, T denotes a binary tree and N is a genetically stable network on the same leaf set as T unless noted otherwise.

4.1 Overview of the Algorithm

A *cherry* is a subtree induced by two sibling leaves ℓ' and ℓ'' and their parent $\alpha_{\ell',\ell''}$, which we denote $\{\alpha_{\ell',\ell''}, \ell', \ell''\}$. It is easy to see that any tree can be transformed into a single node by repeatedly deleting the leaves of a cherry and their incident edges, since this operation turns their parent into a new leaf.

Our algorithm relies on the fact that for any cherry $\{\alpha_{\ell',\ell''}, \ell', \ell''\}$ in T, N displays T if and only if there exists a tree node $p \in \mathcal{T}(N)$ and two disjoint *specific paths* (defined later) $P' \in \mathcal{P}_N(p, \ell')$ and $P'' \in \mathcal{P}_N(p, \ell'')$ such that the modified network $N - [(\mathcal{V}(P') \cup \mathcal{V}(P'')) \setminus \{p\}]$ displays the modified tree $T - \{\ell', \ell''\}$, if we identify leaf p in the modified network with leaf $\alpha_{\ell',\ell''}$ (the parent node of the cherry in T) in the modified tree. Therefore, our algorithm is a recursive procedure which executes the following tasks at each recursive step:

S1: Select a cherry $\{\alpha_{\ell',\ell''}, \ell', \ell''\}$ in T, and determine the corresponding node p and paths P' and P''.

S2: If we fail to find such a node and such paths, N does not display T. Otherwise, recurse on $N - [(\mathcal{V}(P') \cup \mathcal{V}(P'')) \setminus \{p\}]$ and $T - \{\ell', \ell''\}$.

[3] See e.g. the network given at `http://phylnet.info/isiphync/network.php?id=4`

4.2 Three Lemmas

The difficulty in implementing the proposed approach is that a network can display a tree through different subdivisions of the tree and the parent node and edges of a cherry may correspond to different tree nodes and paths in different subdivisions. Therefore, we first prove that the two paths corresponding to the edges of a cherry have special properties.

Lemma 1 (Cherry path). *Let N display T and $\{\alpha_{\ell',\ell''}, \ell', \ell''\}$ be a cherry in T. Then $\alpha_{\ell',\ell''}$ corresponds to a tree node p in each subdivision T' of T in N. Moreover, assume that P' and P'' are the paths in T' that correspond to arcs $(\alpha_{\ell',\ell''}, \ell')$ and $(\alpha_{\ell',\ell''}, \ell'')$, respectively. Then the following properties hold:*

(1) *The node p is not stable on any leaf $\ell \notin \{\ell', \ell''\}$.*
(2) *No vertex in $P' \setminus \{p\}$ is stable on a leaf other than ℓ'.*
(3) *No vertex in $P'' \setminus \{p\}$ is stable on a leaf other than ℓ''.*

In the following discussion, we focus on paths P from an internal node x to a leaf ℓ having the following property:

($\star$) Each $u \in \mathcal{V}(P) \setminus \{x\}$ is either stable only on ℓ or not stable at all.

A path satisfying condition ($\star$) is called a *specific path* (with respect to ℓ). We use $\mathcal{SP}_N(x, \ell)$ to denote the set of specific paths from x to $\ell \in \mathcal{L}(N)$. A path P from u to v is said to be *unstable specific* if no $x \in \mathcal{V}(P) \setminus \{u, v\}$ is stable, where u and v are non-leaf nodes. Note that in a GS network, an unstable specific path is a tree path, since every reticulation is stable. Finally, for a path P and $a, b \in \mathcal{V}(P)$, we use $P[a, b]$ to denote the subpath of P from a to b.

Lemma 2 (Cherry path uniqueness). *Let N be a GS network, $\ell_1, \ell_2 \in \mathcal{L}(N)$, and $a', a'' \in \mathcal{T}(N)$. If there exist two paths $P'_1 \in \mathcal{SP}_N(a', \ell_1)$ and $P'_2 \in \mathcal{SP}_N(a', \ell_2)$ such that $\mathcal{V}(P'_1) \cap \mathcal{V}(P'_2) = \{a'\}$ and two paths $P''_1 \in \mathcal{SP}_N(a'', \ell_1)$ and $P''_2 \in \mathcal{SP}_N(a'', \ell_2)$ such that $\mathcal{V}(P''_1) \cap \mathcal{V}(P''_2) = \{a''\}$, then:*
(1) *Either a'' is a descendant of a' in $P'_1 \cup P'_2$ or vice versa.*
(2) *If a'' is a descendant of a' in P'_2 and u_1 is the highest common node in P'_1 and P''_1 (Figure 2.A), then one of the following facts holds:*
  (a) *$P'_1[a', u_1] = (a', u_1) \in \mathcal{E}(N)$, and $P''_1[a'', u_1]$ is unstable specific.*
  (b) *$P''_1[a'', u_1] = (a'', u_1) \in \mathcal{E}(N)$ and a'' is stable on ℓ_2.*

Proof. (1) Assume the statement is false. Since both P'_i and P''_i end at ℓ_i, they must intersect for $i = 1, 2$. Let u_i be the highest common node in P'_i and P''_i, $i = 1, 2$. Clearly, u_1 and u_2 are reticulations stable on ℓ_1 and ℓ_2, respectively; for each i, the only node common to $P'_i[a', u_i]$ and $P''_i[a'', u_i]$ is u_i (Figure 2.B-D).

If $P'_1[a', u_1], P''_1[a'', u_1], P'_2[a', u_2]$, and $P''_2[a'', u_2]$ are all edges (Figure 2.B), then a' and a'' are the parents of both u_1 and u_2. Since N is GS, either a' or a'' is stable. Clearly a' and a'' are not stable on ℓ_1 and ℓ_2, so stability should involve another leaf ℓ below u_1 or u_2; but this is not possible because there is always a path from a' (resp. a'') to ℓ avoiding a'' (resp. a'). Therefore, one of these four subpaths contains more than one edge. We assume without loss of generality that $P'_1[a', u_1]$ has more than one edge and v and w are the parents of u_1 in P'_1 and P''_1, respectively, where $v \neq a'$. We consider two subcases.

1. If $w = a''$ (Figure 2.C), a'' is clearly not stable on both ℓ_1 and ℓ_2. If a'' is stable on a leaf $\ell \notin \{\ell_1, \ell_2\}$, then ℓ cannot be a descendant of u_1, otherwise the path from a' to ℓ through u_1 would avoid a'', a contradiction. If ℓ is not a descendant of u_1 but is a descendant of the child z of a'' in P_2'', then every path from $\rho(N)$ to ℓ must contain the edge (a'', z) and z. This implies that z is stable on ℓ, contradicting that z is in the specific path P_2''. Therefore, a'' is not stable on any leaf. Since N is GS, v must be stable on a leaf ℓ_3. Since P_1' is a specific path and $v \neq a'$, $\ell_3 = \ell_1$. This implies that v is an ancestor of a'' and so is a', which contradicts the assumption.
2. If $w \neq a''$ (Figure 2.D), either v or w is stable, because N is GS and they are the parents of u_1. Without loss of generality, we may assume w is stable. Since P_1'' is a specific path, w must be stable on ℓ_1. By Proposition 1.(d), w is stable on v, so w either is in $P_1'[a', v]$ or is an ancestor of a'. The former contradicts the fact that u_1 is the highest common node in P_1' and P_1'', whereas the latter implies that a' is a descendant of a'', which contradicts the assumption.

(2) Using the same notation as in (1) (Figure 2.A), since N is GS, either v or w is stable. If v is stable on ℓ_1, by Proposition 1.(d), v is stable on w. So $v = a'$ and $P_1'[a', u_1]$ is just (v, u_1). Let x be in $P_1''[a'', u_1] - \{a''\}$. If x is stable, it must be stable on ℓ_1, since it is in P_1''. This contradicts the fact that there is a path from $\rho(N)$ to ℓ_1 through u_1 avoiding x. Therefore, $P_1''[a'', u_1]$ is unstable specific.

If v is stable on $\ell \neq \ell_1$, then $v = a'$. Otherwise, v would be an internal node of P_1', contradicting the fact that P_1' is in $\mathcal{SP}_N(a', \ell_1)$. If v is not stable, then w must be stable, there are two possible cases. If $w \neq a''$, it must be stable on ℓ_1, as it is in P_1''. Therefore, either w is in $P_1'[a', u_1]$, contradicting that u_1 is the highest common node in P_1' and P_1'', or w is an ancestor of a', contradicting that a' is an ancestor of a''. If $w = a''$, then it is stable on ℓ_2 and $P_1''[a'', u_1]$ is simply (w, u_1). $\square$

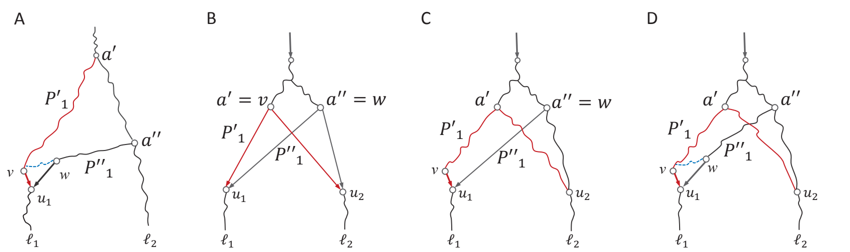

Fig. 2. Illustration of the different cases in the proof of Lemma 2.

Let α_{ℓ_1,ℓ_2} be the parent of ℓ_1 and ℓ_2 in T. Lemma 2.(1) implies that the set of nodes $\{a \mid \exists\ P_1 \in \mathcal{SP}_N(a, \ell_1), P_2 \in \mathcal{SP}_N(a, \ell_2) \text{ s.t. } \mathcal{V}(P_1) \cap \mathcal{V}(P_2) = \{a\}\}$ is totally ordered by the descendant relation, *i.e.* all its elements appear in a path from $\rho(N)$ to ℓ_1. So there is a unique tree node, say p, that is the lowest among all such nodes. Moreover, for any node a in the set, from which there are specific paths P_1 and P_2 going to ℓ_1 and ℓ_2 respectively, Lemma 2.(2) states that if p is a node in P_2, then the path from p to P_1 is an unstable specific path (and vice versa). The next lemma will utilize this property to show that there is a subtree T' of N that is a subdivision of T, in which p corresponds to α_{ℓ_1,ℓ_2}.

Let $t \in \mathcal{T}(N)$. For $\ell_1, \ell_2 \in \mathcal{L}(N)$ and two specific paths whose only common vertex is t, $P_1 \in \mathcal{SP}_N(t, \ell_1)$ and $P_2 \in \mathcal{SP}_N(t, \ell_2)$, we set $N(P_1, P_2)$ to be the subnetwork with vertex set $\mathcal{V}(N)$ and edge set $\mathcal{E}(N) - \{(x,y),(y,x) \mid x \in V(Q) \textit{ and } y \notin V(Q)\} - \{(x,y),(y,x) \mid x \in V(P_1)\backslash\{t\} \textit{ and } y \in V(P_2)\backslash\{t\}\}$ where $Q = (P_1 \cup P_2) \setminus \{t\}$. Note that $N(P_1, P_2)$ is the subnetwork obtained after removing all the edges not in the paths, but incident at some node in Q.

Lemma 3 (Choice of the lower path). *Let N be a GS network and ℓ_1 and ℓ_2 be two sibling leaves in T. Assume that $t \in \mathcal{T}(N)$ and $P_1 \in \mathcal{SP}_N(t, \ell_1)$ and $P_2 \in \mathcal{SP}_N(t, \ell_2)$ are two specific paths whose only common vertex is t such that $N(P_1, P_2)$ displays T. For any path P from u to v in which every $x \in \mathcal{V}(P)\backslash\{u, v\}$ is not stable:*

(1) *If $\mathcal{V}(P) \cap \mathcal{V}(P_j) = \{u\}$ and $\mathcal{V}(P) \cap \mathcal{V}(P_{j'}) = \{v\}$, where $\{j', j\} = \{1, 2\}$, T is also displayed in $N(P_j[u, \ell_j], P[u, v] \cup P_{j'}[v, \ell_{j'}])$.*

(2) *If $\mathcal{V}(P) \cap \mathcal{V}(P_{j'}) = \emptyset$ and $\mathcal{V}(P) \cap \mathcal{V}(P_j) = \{u, v\}$, where $\{j, j'\} = \{1, 2\}$, T is also displayed in $N(P_{j'}, P_j - P_j[u, v] + P[u, v])$.*

Lemma 3.(1) implies that if N displays T, there is a subtree T' that is a subdivision of T, such that p corresponds to α_{ℓ_1,ℓ_2}. The next section includes an algorithm to find the node p.

4.3 The Algorithm

We use two lists at each node u to represent the input network N and the input tree T: the list parent(u) comprises the nodes from which u has an edge, and the list children(u) consists of nodes to which u has an edge.

We say that a node u is *reachable* from the network root if there is a path from the root to u. Using a breadth-first search, we can determine the sets of descendants for each vertex in $O(|\mathcal{E}(N)| + |\mathcal{V}(N)|)$ time. To determine the stability of a node u, one can compute the set $R_{not}(u)$ of leaves that are reachable from the root in $N - u$. Obviously, the set $\mathcal{PDL}_N(u)$ of nodes on which u is stable is $\mathcal{L}(N) - R_{not}(u)$, so u is stable if and only if $R_{not}(u) \neq \mathcal{L}(N)$. Therefore, we can determine whether or not a node is stable on a leaf in time $O(|\mathcal{E}(N)| + |\mathcal{V}(N)|)$.

We first find two sibling leaves l_1 and l_2 with parent α_{l_1,l_2}, which takes $O(|\mathcal{L}(T)|)$ time. We then extend a specific path starting at l_1 by moving a node up each time to find a $p \in \mathcal{T}(N)$ such that if N displays T, there is a subdivision of T in which p corresponds to α_{l_1,l_2}. Assume we arrive at a node w. If w is stable on a leaf $z \notin \{l_1, l_2\}$, then we conclude that N does not display T. If w is stable on l_2, or if there is a specific path from w to l_2, then w must be p if N displays T and we are done, so we continue our analysis by assuming otherwise.

If w is a tree node, we simply move up to its unique parent $p(w)$. If w is a reticulation, it is stable on l_1. Let $p(w) = \{u, v\}$. We have to chose either u or v to move up using the stability property that w is only stable on l_1 and at least one of u and v is stable. By Proposition 1.(c), u and v cannot both be stable on l_1. If u is stable on l_1 and v is stable on l_2, by Proposition 1.(d), u must also be stable on l_2. Therefore, we just need to consider eight different conditions (Table 1) to choose u or v to move up.

Table 1. When w is stable on l_1, there are six combinations of its parents u and v for consideration. Here, $l(u, v) = u$ if u is a descendant of v, or v otherwise

Cond. S/N	**Stability of u**	**Stability of v**	**Selection**
C1	$\mathcal{PDL}_N(u)\backslash\{l_1, l_2\} \neq \emptyset$	$\mathcal{PDL}_N(v)\backslash\{l_1, l_2\} \neq \emptyset$	Neither
C2	$\mathcal{PDL}_N(u)\backslash\{l_1, l_2\} \neq \emptyset$	$\mathcal{PDL}_N(v) \subseteq \{l_1, l_2\}$	v
C3	u is stable on l_1 (and eventually on l_2)	v is not stable	v
C4	u is not stable	v is stable only on l_2	v
C5	$\mathcal{PDL}_N(u) \subseteq \{l_1, l_2\}$	$\mathcal{PDL}_N(v)\backslash\{l_1, l_2\} \neq \emptyset$	u
C6	u is not stable	v is stable on l_1 (and eventually on l_2)	u
C7	u is stable only on l_2	v is not stable	u
C8	u is stable on l_2	v is stable on l_2	$l(u, v)$

If condition C1 holds, w cannot be a node in the path corresponding to the edge (α_{l_1,l_2}, l_1) in any subdivision T' of T. This is because a leaf in either $\mathcal{PDL}_N(u) \setminus \{l_1, l_2\}$ or $\mathcal{PDL}_N(v) \setminus \{l_1, l_2\}$ will not appear in any T' that can be contracted into a tree in which l_1 and l_2 are siblings. Similarly, if C2 holds, u cannot be a node in the path corresponding to the edge (α_{l_1,l_2}, l_1) in a subdivision T' of T. Therefore, we select v. If C3 holds, since u and w are stable on l_1, by Proposition 1.(d), u is stable on v and we move to v if v is not stable. If C4 holds, by Proposition 1.(e), if u is below v, there is a reticulation r' such that there is a tree path from r' to u, r' is not above l_2, and r' is below v. This implies that r' is stable on a leaf other than l and l_2, so we choose v. If u is not below v, then we also choose v because we need to choose the lower one. Conditions C5–C7 are symmetric to C2–C4 and so we select u to move up if they are true. If C8 holds, then, u is a descendant of v or vice versa. Clearly, we have to choose whichever is lower than the other. Algorithm 1 summarizes the whole procedure.

As we have seen, the property that each reticulation has a stable parent is crucial in enabling a correct choice at a reticulation stable on a leaf under consideration. A simple condition allows us to determine whether we have reached p while moving up from x: there is a unstable specific tree path from p to l_2 or to a reticulation stable on l_2, because there is a specific path from p to l_2. Thus, we obtain Algorithm 2 to solve the TC problem.

Algorithm 1: Move up one node to find p

Procedure *MoveUpInSpecificPath(w, l_1, l_2, P, N)*

  Input: node w, leaves l_1 and l_2 and path P in network N

  Output: **false** if N does not display T, **true** if no final decision can yet be made

  if *w is a tree node* **then**

    $P \leftarrow P \cup \{w\}$; $N \leftarrow N - (\text{parent}(w), w)$; $w \leftarrow \text{parent}(w)$;

  `// Select a parent at a reticulation`

  if *w is a reticulation stable on l_1 with parents $\{u, v\}$* **then**

    if *C1* **then**

      return false;

    if *C2 or C3 or C4 or (C8 and v is lower)* **then**

      $P \leftarrow P \cup \{w\}$; $N \leftarrow N - (u, w)$; $w \leftarrow v$;

    if *C5 or C6 or C7 or (C8 and u is lower)* **then**

      $P \leftarrow P \cup \{w\}$; $N \leftarrow N - (v, w)$; $w \leftarrow u$;

    return true;

  else

    return false;

Theorem 1. *Algorithm 2 solves the TC problem for GS networks in quadratic time.*

Proof. Assume the input network N displays the input tree T, and let $\mathcal{SD}_N(T)$ be the set of subdivisions of T in N. Let α_{ℓ_1,ℓ_2} be the parent of the sibling leaves ℓ_1 and ℓ_2 in T selected in line 3 of Algorithm 2. Recall that by Lemma 2, the set $\{a \mid \exists P_1 \in \mathcal{SP}(a, \ell_1), P_2 \in \mathcal{SP}(a, \ell_2) \text{ s.t. } \mathcal{V}(P_1) \cap \mathcal{V}(P_2) = \{a\}\}$ has a lowest element p. If N displays T, by Lemma 3, p must correspond to α_{ℓ_1,ℓ_2} in some subdivision of T in N. We now show that Algorithm 2 correctly finds p.

Let P_i be the path from p to ℓ_i in a subdivision $T' \in \mathcal{SD}_N(T)$ corresponding to the edge $(\{\alpha_{\ell_1,\ell_2}, \ell_1, \ell_2\}, \ell_i)$ in the cherry in T for $i = 1, 2$. By Lemma 1, P_1 and P_2 are specific paths. Let us prove that the first while-loop exits at $w_1 = p$. Assume t is the last vertex in P_1 at which the algorithm has moved off during the first while-loop before stopping at $w_1 = w \neq p$ (Figure 3.A). So t is a reticulation with a parent v in P_1 and the other parent u to which the algorithm moved from t. Let P be the path consisting of all vertices visited by the algorithm after t.

Since t is a reticulation in P_1, it is stable on ℓ_1. By the definition of the moving up procedure MoveUpInSpecificPath, moving from t to u implies that C5, C6, C7 or C8 holds. C5 cannot be true, as v is in P_1 and cannot be stable on a leaf not equal to ℓ_1. If C8 holds, then $v = p$. u is not in P_2, otherwise u is lower than p and there are specific paths from u to ℓ_1, ℓ_2. If u is not in P_2, then it is above p since it is stable on ℓ_2, but then it is also above v, contradicting that we choose u. If C7 is valid, then the algorithm should stop at u, as u is stable on ℓ_2, implying $w = u$. This is impossible as w is not in P_2. If C6 is valid, then v is stable on ℓ_1 and u is not stable. By Proposition 1.(d), v is stable on u, which implies that v is an ancestor of w or vice versa.

1. If node v is an ancestor of w (Figure 3.B), then P can be extended into a path $\overline{P}$ from v to u. Since v is stable on ℓ_1, there are no reticulations in $\overline{P}[v, w]$. Furthermore, no node in $\overline{P}[v, w]$ is stable on a leaf, since the first edge of $\overline{P}$ is not in T'. Otherwise, if a node y in $\overline{P}[v, w]$ is stable on ℓ, y is not in T', and then ℓ is not in T', contradiction. That the algorithm stopped at w implies that (i) w is a reticulation with both parents being stable on a leaf not in $\{\ell_1, \ell_2\}$, or (ii) there is an unstable specific path P' from w to a node s that is stable on ℓ_2.
   Case (i) is not true, because we have observed that the parent of w in $\overline{P}$ is not stable. If case (ii) is true, s must be in P_2. We have another pair of specific paths $P[w, t] \cup P_1[t, \ell_1]$ and $P'[w, s] \cup P_2[s, \ell_2]$, which is impossible because w is not in $P_1 \cup P_2$ (Lemma 2.(1)).
2. If node w is an ancestor of v (Figure 3.C), then since v is stable on ℓ_1, the path P taken by the algorithm from u to w must go through v, contradicting the choice of t. Using an argument similar to the one presented above, we can show that the second while-loop stops at p correctly . After the execution of the two while loops, we have that $w_1 = w_2 = p$. By Lemma 3, the recursive call in Step 3 is correct.

This shows that if N displays T, our algorithm finds the lowest image of the parent of ℓ_1 and ℓ_2 together with specific paths P_1 and P_2 in a subdivision of T. By Lemma 3, N displays T if and only if $T - \ell_1 - \ell_2$ is displayed in $N - P_1 - P_2$. This concludes the proof of correctness of the algorithm.

Regarding the time complexity of the algorithm: note that each recursive step removes two sibling leaves from the input tree and that N has at most $|\mathcal{E}(N)| = O(|\mathcal{L}(N)|)$ nodes (see [3]). In different recursive steps, the nodes whose stability is examined are different, and the time spent on checking stability is at most $|\mathcal{V}(N)| \times O(|\mathcal{E}(N)| + |\mathcal{V}(N)|) = O(|\mathcal{L}(N)|^2)$. Before entering the next recursive step, the nodes that have been visited in the current step are removed. Therefore, the algorithm has quadratic time complexity. □

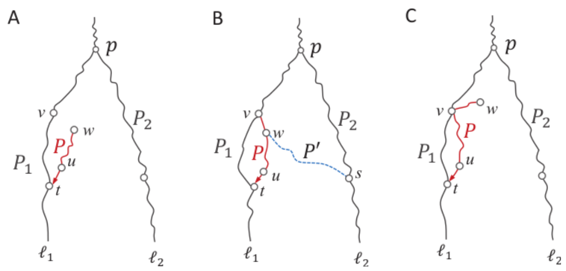

Fig. 3. Illustration for the proof of Theorem 1.

Algorithm 2: Deciding whether a given GS network displays a given tree.

Procedure *Tree-Display(N, T)*

Input: a GS network N with information on stability, a tree T

Output: **true** if N displays T, **false** otherwise

if T *is a single node* **then**
    return true;

Compute a cherry $\{\alpha_{\ell_1,\ell_2}, \ell_1, \ell_2\}$ in T;

$w_1 \leftarrow \text{parent}(\ell_1)$; $P_1 \leftarrow \{\ell_1, w_1\}$;   `// Initialize to start with` ℓ_1

`/* Move up to reach the lowest` p `corresponding to` α_{ℓ_1,ℓ_2} `in a subdivision of` T `*/`

while *no unstable specific path from* w_1 *to* ℓ_2 *or a node stable on* ℓ_2 **do**
    if *MoveUpInSpecificPath*$(w_1, \ell_1, \ell_2, P_1, N)$ = **false then**
        return false;

$w_2 \leftarrow \text{parent}(\ell_2)$; $P_2 \leftarrow \{\ell_2, w_2\}$;   `// Initialize to move up at` ℓ_2

while $w_2 \neq w_1$ *and* w_2 *is below* w_1 **do**
    if *MoveUpInSpecificPath*$(w_2, \ell_2, \ell_1, P_2, N)$ = **false then**
        return false;

if $w_2 \neq w_1$ **then**
    return false;

return *Tree-Display*$(N - P_1 - P_2, T - \ell_1 - \ell_2)$;

5 Conclusion

In the present work, we introduced the class of GS networks to study the TC problem. In [3], we developed a quadratic-time algorithm for nearly stable networks by iteratively selecting an edge entering a reticulation to delete in the end of a longest path in a nearly stable network. Here, using a different approach, we have proved that the TC problem can also be solved in quadratic time for GS networks.

A trivial $2^{|\mathcal{R}(N)|} \cdot \text{poly}(|\mathcal{L}(N)|)$ algorithm solves the TC problem as follows: for each reticulation, simply guess which entering edge to delete. However, the number of reticulations can be quite large e.g. in the case of bacterial genomes [9], and many gene families need to be examined. Therefore, our proposed algorithm with low time complexity is definitely valuable for model verification in comparative genomics.

Several problems remain open for future study. First, Figure 4 summarizes the inclusion relationships between the network classes defined in this paper and other well-studied classes defined in [5]. Galled networks are a generalization of level-1 networks (also called galled trees), comprising a subclass of stable networks [5]. The complexity of the TC problem for galled networks is open.

Second, a natural generalisation of the TC problem is to decide whether a given network displays another given network. Is it possible to determine in polynomial time whether a given GS network displays another given one?

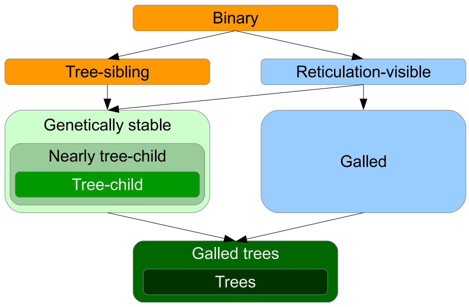

Fig. 4. Inclusion relationships between GS networks and other classes, represented by colored rectangles. A class that is drawn within another one is a subclass of the latter; an arrow points from a nested class cluster to another if classes in the former are all a superclass of the classes in the latter. A network is tree-child if every node in it has a child that is a tree node.

6 Acknowledgments

The project was financially supported by Merlion Programme 2013.

References

1. Chan, J. M., et al. (2013) Topology of viral evolution. *PNAS*, 110, 18566-18571.
2. Cordue, P., Linz, S., Semple, C. (2014). Phylogenetic networks that display a tree twice. *Bulletin Math. Biol.*, 76, 2664-2679.
3. Gambette, P., et al. (2015) Locating a tree in a phylogenetic network in quadratic time, In *Proc. of RECOMB'2015*, pp. 96-107. Springer.
4. Gusfield, D. (2014) *ReCombinatorics: The Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks*. MIT Press, Cambridge, USA.
5. Huson, D. H., Rupp, R., Scornavacca, C. (2010) *Phylogenetic Networks*. Cambridge University Press, Cambridge, UK.
6. van Iersel, L., Semple, C., Steel, M. (2010) Locating a tree in a phylogenetic network. *Inform. Proces. Lett.*, 110, 1037-1043.
7. Kanj, I. A., Nakhleh, L., Than, C., Xia, G. (2008) Seeing the trees and their branches in the network is hard. *Theoret. Comput. Sci.*, 401, 153-164.
8. Nakhleh, L. (2013) Computational approaches to species phylogeny inference and gene tree reconciliation. *Trends Ecol. Evol.*, 28, 719-728.
9. Treangen, T. J., Rocha, E. P. (2011) Horizontal transfer, not duplication, drives the expansion of protein families in prokaryotes. *PLOS Genetics*, 7, e1001284.
10. Wang, L., Zhang, K., Zhang, L. (2001) Perfect phylogenetic networks with recombination. *J. Comput. Biol.*, 8, 69-78.

Appendix: Omitted Proofs

Proof (Proposition 1).

(a) See Proposition 1.(3) in [3].
(b) It clearly follows from (a).
(c) It follows from the fact that there is a path from the root to r avoiding at least one parent of r.
(d) Let u and r be stable on a leaf ℓ. For a path P from r to ℓ, the concatenation of the edge (v, r) and P produces a path avoiding u. Therefore, any path from $\rho(N)$ to v must go through u, implying that u is stable on v.
(e) If u is in a path between v and ℓ', the stability of v on ℓ' implies that every path from $\rho(N)$ to u must go through v and therefore v is also stable on ℓ, a contradiction.
Assume z is a node between v and ℓ'. If there is a tree path from z to u, then every path P from $\rho(N)$ to u must pass through z. If P does not pass through v, the subpath of P from $\rho(N)$ to z can be extended into a path from $\rho(N)$ to ℓ' that does not go through v, contradicting our assumption that v is stable on ℓ'. Therefore, we have shown that v is stable on u and therefore on ℓ, which is impossible and implies that the second statement in (e) is true. □

Proof (Proposition 2). Let N be a GS network and r be a reticulation in N with parents p_1 and p_2. Since N is GS, either p_1 or p_2 is stable, so we assume wlog that p_1 is stable. By Proposition 1.(a), p_1 must have another child that is a tree node, and N is therefore a tree-sibling network. □

Proof (Lemma 1). Let T' be a subdivision of T in N and $\alpha_{\ell',\ell''}$ correspond to $p \in \mathcal{V}(T')$. Recall that T' is obtained by removing an incoming edge at each reticulation. Each $r \in \mathcal{R}(N)$ becomes a degree-2 node in T' if it is in T'. Thus, p is a tree node in T'.

(1) Let $\ell \notin \{\ell', \ell''\}$ be a leaf in N. Since $\rho(T')$ is identical to $\rho(N)$, there is a unique path X from $\rho(N)$ to ℓ in T'. Since p corresponds to $\alpha_{\ell',\ell''}$, ℓ is not below p in T' and thus X does not pass through p. Therefore, p is not stable on ℓ.
(2) Let $P' = (u_{k+1} = p, u_k, \ldots, u_1, u_0 = \ell')$. Suppose on the contrary that u_j is stable on some leaf $\ell \neq \ell'$ for some $1 \leq j \leq k$. Then in T', ℓ must be a descendant of u_j, implying that u_j is a common ancestor of ℓ and ℓ' in T'. This contradicts our assumption that ℓ' and ℓ'' belong to a cherry in T.
(3) The proof is similar to that of case (2). □

Proof (Lemma 3).

(1) Without loss of generality, we assume that $j' = 1$ and $j = 2$ (Figure 5.A). Note that $P[u, v] \cup P_1[v, \ell_1]$ is a path from u to ℓ_1. Assume T' is a subdivision of T in $N(P_1, P_2)$. Since P_1 and P_2 are the unique path from t to ℓ_1 and ℓ_2 in T', respectively, and every node in P has degree-2 in $N(P_1, P_2)$, t is

the node corresponding to the parent of ℓ_1 and ℓ_2 in the display T' of T (Figure 5.A).

Since v is a reticulation in P_1, it is stable only on ℓ_1. There are two cases for consideration. Let $N'' = N(P_2[u, \ell_2], P[u, v] \cup P_1[v, \ell_1])$.

If $P[u, v]$ is the edge (u, v), define $T'' = T' - \mathcal{E}(P_1[t, v]) + (u, v)$. Clearly, T'' is a subdivision of T in N''. Therefore T is displayed in N''.

If P contains more than one edge, there are no reticulations other than v in $P[u, v]$, as any reticulation is stable. By definition, the first and last edge of P are not in $N(P_1, P_2)$. Since P does not contains any reticulations, all the branches and nodes of P are not in T'. Therefore, $T'' = T' - P_1[t, v] + P$ is a subtree in N''. Clearly, T is a contraction of T'', in which the parent of ℓ_1 and ℓ_2 corresponds to u.

(2) Without loss of generality, we may assume that $j = 2$ and T' is a subdivision of T in $N(P_1, P_2)$. Since v is a reticulation in P_2, v is stable only on ℓ_2. Since there is no stable node in P except for v, all nodes other than v are tree nodes. Since the first and last edges are not in $N(P_1, P_2)$, all the nodes other than v are not in T'. Therefore, $T'' = T' - P_2[u, v] + P$ is a subdivision of T in $N(P_1, P_2 - P[u, v] + P)$. □

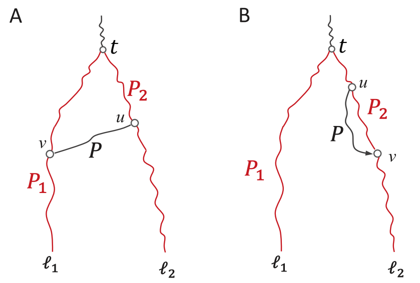

Fig. 5. Illustration for the proof of Lemma 3.